НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Тарасенко В.П.
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050102 «Комп'ютерна інженерія»

на тему: «Веб-орієнтована система для визначення збігів в аудіофайлах»

Виконала: студентка IV курсу, групи КВ-52

Грабовська Ліза Русланівна
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н., доцент Сапсай Т.Г. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студентка_____

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
«___»_____2019 р.

ЗАВДАННЯ
на дипломний проект студентки
Грабовської Лізи Русланівни

1. Тема проекту: веб-орієнтована система для визначення збігів в аудіофайлах, керівник проекту Сапсай Тетяна Григорівна, доц. каф. СПіСКС, к.т.н., затверджені наказом по університету від «22» травня 2019 р. №1330-С

2. Термін подання студентом проекту “13” червня 2019 р.

3. Вихідні дані до проекту: див. технічне завдання.

4. Зміст пояснювальної записки: аналіз існуючих рішень та обґрунтування теми дипломного проекту, вибір програмних засобів для реалізації проекту, система визначення збігів в аудіофайлах.

5. Перелік графічного матеріалу:

- алгоритм динамічного пошуку найбільшої загальної підпослідовності знизу вгору. Схема алгоритму;
- алгоритм декодування Хаффмана для аудіофайлу. Схема алгоритму;
- створення акустичного відбитку аудіофайлу. Схема алгоритму;
- архітектура системи. Схема структурна.

6. Консультанти розділів проекту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н.		

7. Дата видачі завдання «25» жовтня 2019р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.12.2018
2.	Розроблення та узгодження технічного завдання	10.02.2019
3.	Аналіз існуючих рішень ідентифікації аудіофайлів	15.02.2019
4.	Побудова архітектури системи	30.04.2019
5.	Вибір програмних засобів для реалізації проекту	10.05.2019
6.	Розробка системи та перевірка роботи програмного продукту	15.05.2019
7.	Підготовка пояснювальної записки та графічної частини дипломного проекту	20.05.2019
8.	Оформлення документації дипломного проекту	23.05.2019
9.	Попередній огляд матеріалів дипломного проекту на кафедрі	29.05.2019

Студент _____ Грабовська Л.Р.
Керівник проекту _____ Сапсай Т.Г.

* Консультантом не може бути зазначено керівника дипломного проекту.

АНОТАЦІЯ

Дипломний проект включає пояснювальну записку (54 стр., 10 рис., 8 табл.).

Об’єктом розробки є веб-орієнтована система для визначення збігів в аудіофайлах.

Метою даного проекту є розробка веб-орієнтованої системи для визначення збігів в аудіофайлах. Дослідження технології створення акустичних відбитків.

В ході виконання дипломного проекту:

- проведено аналіз аналогів програмного забезпечення;
- розглянуто існуючі алгоритми ідентифікації аудіофайлів;
- досліджено методи декодування аудіофайлів;
- досліджено технологію створення акустичних відбитків аудіофайлів;
- розроблено архітектуру системи;
- розроблено веб-додаток для порівняння завантажених користувачем аудіофайлів.

Ключові слова: збіги в аудіо файлах, веб-орієнтована система, JavaScript, Python, React, акустичний відбиток, AudioRead, Chromaprint.

ABSTRACT

This diploma project includes an explanatory note (54 p., 10 figures, 8 tables).

The object of the development is a web-based system for identifying matches in audiofiles.

The purpose of this project is to develop a web-based system for identifying matches in audiofiles. Research of acoustic fingerprint technology.

During the implementation of the graduation project:

- analyzed software analogues;
- reviewed existing audio identification algorithms;
- investigated methods of decoding audio files;
- investigated the technology of acoustic imprinting of audio files;
- developed the architecture of the system;
- developed the web application to compare downloaded audiofiles.

Keywords: matches in audio files, web-oriented system, JavaScript, Python, React, acoustic fingerprint, AudioRead, Chromaprint.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ.045490.002 ТЗ	Веб-орієнтована система	4		
			визначення збігів в			
			аудіофайлах.			
			Технічне завдання			
	А4	ІАЛЦ.045490.003 ТП	Веб-орієнтована система	2		
			визначення збігів в			
			аудіофайлах.			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045490.004 ПЗ	Веб-орієнтована система	54		
			визначення збігів в			
			аудіофайлах.			
			Пояснювальна записка			

Змін.	Арк.	№ докум.	Підпис	Дата	**ІАЛЦ. 045490.001 ОА**		
Розробив		Грабовська Л.Р.			*Веб-орієнтована система для визначення збігів в аудіофайлах* **Опис Альбому**	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Сапсай Т.Г.					
Н. контроль		Клятченко Я.М.				***КПІ ім. Ігоря Сікорського, ФПМ, КВ-52***	
Затвердив		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.005 Д1	Веб-орієнтована система	1		
			визначення збігів в			
			аудіофайлах.			
			Алгоритм динамічного			
			пошуку найбільшої			
			загальної			
			підпослідовності знизу			
			вгору.			
			Схема алгоритму			
	А4	ІАЛЦ.045490.006 Д2	Веб-орієнтована система	1		
			визначення збігів в			
			аудіофайлах.			
			Алгоритм декодування			
			Хаффмана для			
			аудіофайлу.			
			Схема алгоритму			
	А4	ІАЛЦ.045490.007 Д3	Веб-орієнтована система	1		
			визначення збігів в			
			аудіофайлах.			
			Створення акустичного			
			відбитку аудіофайлу			
			Схема алгоритму			

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.008 Д4	Веб-орієнтована система	1		
			визначення збігів в			
			аудіофайлах.			
			Архітектура системи.			
			Схема структурна			
		Диск CD-ROM	Матеріали дипломного	1		
			проекту			

Змін.	Арк.	№ докум.	Підпис	Дата	***ІАЛЦ. 045490.001 ОА***	Арк. 3

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: «Веб-орієнтована система для визначення збігів в аудіофайлах».

Галузь застосування: використання в якості програмного забезпечення для визначення збігів в аудіофайлах, визначених користувачем.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ

Метою даної роботи є вивчення теорії декодування аудіофайлів, створення акустичних відбитків, розробка програмного забезпечення для визначення збігів в аудіофайлах, для використання кінцевими користувачами.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами інформації є технічна та науково-технічна література, технічна документація використаних технологій, публікації у періодичних виданнях та електронні статті у мережі Інтернет, що стосуються даної проблеми.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до апаратного забезпечення:

- процесор з тактовою частотою не менше 800 Гц;
- оперативна пам’ять: від 1024 Мб;
- наявність доступу до мережі Internet.

5.2. Вимоги до програмного забезпечення:

- браузер Google Chrome;
- бібліотека FFmpeg.

5.3. Вимоги до програмного продукту, що розробляється:

- забезпечити зручний для користувача інтерфейс;
- можливість порівняння аудіофайлів;
- робота в режимі on-line;
- підтримка популярних браузерів;
- можливість завантаження та обробки стиснених звукових файлів;
- створення акустичного відбитку файлу.

6. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

• пояснювальна записка;

• креслення:

- «Алгоритм динамічного пошуку найбільшої загальної підпослідовності знизу вгору. Схема алгоритму»;
- «Алгоритм декодування Хаффмана для аудіофайлу. Схема алгоритму»;

- «Створення акустичного відбитку аудіофайлу. Схема алгоритму»;
- «Архітектура системи. Схема структурна».

7. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.12.2018
2.	Розроблення та узгодження технічного завдання	10.02.2019
3.	Аналіз існуючих рішень ідентифікації аудіофайлів	15.02.2019
4.	Побудова архітектури системи	30.04.2019
5.	Вибір програмних засобів для реалізації проекту	10.05.2019
6.	Розробка системи та перевірка роботи програмного продукту	15.05.2019
7.	Підготовка пояснювальної записки та графічної частини дипломного проекту	20.05.2019
8.	Оформлення документації дипломного проекту	23.05.2019
9.	Попередній огляд матеріалів дипломного проекту на кафедрі	29.05.2019

№ з/п.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
1	А4	ІАЛЦ.045490.004	Веб-орієнтована система визначення збігів в аудіофайлах. Пояснювальна записка	54		
2	А4	ІАЛЦ.045490.005 Д1	Веб-орієнтована система визначення збігів в аудіофайлах. Алгоритм динамічного пошуку найбільшої загальної загальної підпослідовності підпослідовності знизу вгору. Схема алгоритму	1		

Зм.	Арк.	№ докум.	Підп.	Дата	***ІАЛЦ. 045490.003 ТП***		
Розроб.		Грабовська Л.Р.			*Веб-орієнтована система для визначення збігів в аудіофайлах* **Відомість технічного проекту**	**Літ.**	**Аркуш** 1 / **Аркуші** 2
Перев.		Сапсай Т.Г.					
Н. контр.		Клятченко Я.М.				***КПІ ім. Ігоря Сікорського, ФПМ, КВ-52***	
Затв.		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
3	А4	ІАЛЦ.045490.006 Д2	Веб-орієнтована система	1		
			визначення збігів в			
			аудіофайлах.			
			Алгоритм декодування			
			Хаффмана для			
			аудіофайлу.			
			Схема алгоритму			
4	А4	ІАЛЦ.045490.007 Д3	Веб-орієнтована система	1		
			визначення збігів в			
			аудіофайлах.			
			Створення акустичного			
			відбитку аудіофайлу			
			Схема алгоритму			
5	А4	ІАЛЦ.045490.008 Д4	Веб-орієнтована система	1		
			визначення збігів в			
			аудіофайлах.			
			Архітектура системи.			
			Схема структурна			
6		Диск CD-ROM	Матеріали дипломного проекту.	1		

ЗМІСТ

ДОДАТКИ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

AAC – патентований формат аудіофайлу з втратами.

API – програмний інтерфейс додатку, включає в себе набір протоколів, підпрограм і інструментів для створення зовнішніх додатків.

Apple Music – сервіс потокової музики компанії Apple.

DOM – інтерфейс прикладного програмування, який розглядає документ XML як деревоподібну структуру.

IE – браузер компанії Microsoft.

IFPI – міжнародна федерації виробників фонограм

JSX – розширення синтаксису JavaScript.

MP3 – формат файлу для зберігання аудіоінформації.

Now Playing – додаток для ідентифікації музичних творів від компанії Google.

ODG – параметр, що використовується для оцінки алгоритму кодування аудіофайлів.

PCM – не стиснений аудіосигнал.

PEAQ – стандартизований алгоритм для об'єктивного вимірювання якості звуку.

Shazam – додаток для ідентифікації музичних творів.

SNR – відношення потужності сигналу до потужності шуму.

Sound Search – технологія для ідентифікації музичних творів компанії Google.

Spotify – сервіс потокової музики.

SVM – лінійний алгоритм, що використовується в задачах класифікації та регресії. Алгоритм створює лінію або гіперплощину, яка розділяє дані на класи.

WMA – аудіокодеки та відповідні формати аудіокодування, розроблені компанією Microsoft.

ПК – персональний комп'ютер.

ВСТУП

Проблема знаходження збігів в аудіофайлах є актуальною для багатьох напрямів наукової і практичної діяльності: для біологів, що вивчають звуки тварин, для широкого спектру сервісів, що надають можливість збереження аудіоінформації. Можливість досить точно знаходити однакові (або дуже схожі) аудіозаписи, вирішує ряд проблем, наприклад:

- дублікації одного й того ж треку під різними назвами;
- пошук композиції з різною бітовою швидкістю;
- додавання обкладинки і тексту до всіх варіантів музичного твору;
- удосконалення механізму рекомендацій аудіофайлів користувачам;
- поліпшення роботи зі скаргами власників контенту.

Завданням даного дипломного проекту є вивчення та дослідження теорії декодування аудіофайлів, обчислення частотних характеристик аудіосигналів, створення акустичних відбитків та розробка веб-орієнтованої системи для визначення збігів в аудіофайлах, що має відповідати таким вимогам:

- зручності користування програмою;
- застосування сучасних технологій та новітніх рішень;
- роботи в режимі on-line;
- можливості подальшого розвитку програми.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1. Аналіз існуючих рішень

Найбільш відомим представником рішення порівняння та ідентифікації аудіофайлів є додаток Shazam, один з найпопулярніших додатків в світі, яким щомісяця користуються більше 100 мільйонів людей [1]. Маючи в якості вхідних даних 20 секунд звучання музичної композиції, неважливо, чи буде це частина вступу, приспіву або основного мотиву музичного твору, додаток Shazam створить сигнатурний код, звіриться з базою даних і завдяки власному алгоритму розпізнавання музики визначить назву музичного твору.

Переваги додатку Shazam:

- простий у використанні інтерфейс;
- функція Auto Shazam для автоматичного ідентифікації пісень;
- інтеграція з голосовим помічником Siri;
- зручний пошук музики;
- безкоштовне розповсюдження та використання.

Недоліки додатку Shazam:

- для пошуку може бути використаний лише оригінальний трек, без додаткового шуму;
- немає можливості збереження плейлистів.

Компанія Google представила додаток Now Playing, який здатен розпізнавати музику, що звучить у фоновому режимі пристрою користувача [2]. Now Playing постійно аналізує навколишні звуки, не вимагає підключення до Інтернету, вся база треків зберігається на пристрої користувача. Основною особливістю нового додатку стало використання нейромереж для ідентифікації треків. Для розпізнавання додаток Now Playing використовує акустичний відбиток запису звуків навколишнього простору, порівнюючи його з відбитками, які знаходяться в

пристрою користувача. У додатку Now Playing зберігається тисячі пісень, база даних постійно оновлюється, додаються нові треки, а непопулярні видаляються. Це створює деякі обмеження швидкості та точності розпізнавання аудіофайлів.

Проте компанія Google покращила свій додаток розпізнавання звуку за допомогою технології Sound Search. Алгоритм, на основі реалізовано даний додаток, обчислює унікальні відбитки аудіосигналу кожні 0,5 секунди тоді як в Shazam на це витрачається 1 секунда. Компанія Google також збільшила в чотири рази розмір нейронної мережі, яка порівнює унікальні акустичні відбитки. В результаті чого, додаток Now Playing ідентифікує аудіофайл швидше у порівнянні з Shazam.

Переваги додатку Now Playing:

- для пошуку може бути використаний трек з додатковим шумом;
- дозволяє збереження вибраних треків;
- швидкий результат ідентифікації треку.

Недоліки технології Now Playing:

- можливий помилковий результат для треку з додатковим шумом;
- безкоштовна версія містить рекламу.

Таблиця 1.1

Таблиця порівняння додатків Now Playing і Shazam

	Now Playing	Shazam
Вартість	Безкоштовно, існує окрема платна преміум-версія	Безкоштовно
Сервіси потокової музики, що підтримуються додатком	Apple Music і Spotify	Apple Music і Spotify
Пошук музики	Так	Так
Збереження вибраних пісень	Так	Ні

1.2. Основні відмінності рішення, що запропоновані в дипломному проекті від існуючих рішень

Основною відмінністю запропонованого рішення є наявність веб-інтерфейсу, що надає змогу користуватися додатком на ПК та мобільних пристроях. Оскільки даний додаток порівнює два завантажені користувачем аудіофайли, це дає змогу порівнювати не лише пісні, а й інші музичні твори, різні звукові треки, наприклад, записи звуків птахів. Крім того присутній інтерфейс API, який може бути використане для побудови інших додатків, що базуються на створенні акустичних відбитків аудіофайлів, та їх порівнянні.

1.3. Аналіз способів ідентифіації аудіофайлів

Існує декілька алгоритмів обробки аудіофайлів, що можуть бути використані для ідентифікації та класифікації інформації даного типу:

- акустичний відбиток;
- водяний знак аудіо;
- SVM;
- момент Черніке;
- комп'ютерний зір.

Акустичний відбиток – це представлення аудіосигналу у вигляді набору значень параметрів, що описують фізичні властивості аудіосигналу. Надійний алгоритм акустичного відбитку повинен враховувати перцептивні характеристики звуку. Якщо два файли звучать однаково для людського вуха, акустичні відбитки таких файлів повинні збігатися, навіть якщо їхні двійкові зображення абсолютно різні.

Акустичні відбитки не є хеш-функціями, чутливими до будь-яких невеликих змін у даних, вони більш аналогічні відбиткам пальців людини, де невеликі відхилення, які незначні для особливостей використання відбитків пальців, допускаються.

Більшість методів звукового стиснення (AAC, MP3, WMA, Vorbis) вносять радикальні зміни в бінарне кодування аудіофайлу, не впливаючи на сприйняття аудіоінформації людським вухом. Алгоритм має забезпечувати ідентифікацію запису після того, як він пройшов через таке стиснення, навіть якщо якість звуку значно знижено.

Технології аудіовідбитку наразі привертають увагу, оскільки вони дозволяють ідентифікувати звук незалежно від його формату і без необхідності додавання метаданих або водяних знаків до файлу. Успішне розпізнавання на основі

відбитків можливе лише при унікальних аудіо фрагментах. Однак у реальному контенті можуть бути аудіо-дублікати, наприклад, однакова фонова музика.

Водяний знак аудіо – це якесь повідомлення, яке вбудоване в аудіооб'єкт під час запису [3]. Водяний знак є ідентифікаційним знаком, який може бути використаний для підтвердження автентичності, ідентифікації.

Інформацію про трек можна отримати, виділивши вбудоване повідомлення в аудіоінформацію. Вбудовуваний водяний знак має не змінювати сприйняття аудіоінформації людиною, тобто він має бути прозорим.

Достовірною кількісною характеристикою прозорості є параметр SNR, що визначає відношення потужності вихідного сигналу до потужності спотворень, викликаних водяним знаком. У відповідності до рекомендацій IFPI параметр SNR за значенням повинен бути більш 20 дБ. Поруч з SNR для оцінки прозорості використовується параметр ODG, що розраховується у відповідності до алгоритму PEAQ, та змінюється від 0, при абсолютній непомітності, до мінус 4 - при виявленні спотворень, що викликають помітні для людського слухового апарату, зміни в аудіоінформації. На відмінну від SNR, параметр ODG враховує особливості побудови слухової системи людини.

Використання цього методу не дозволяє визначити два однакових аудіооб'єкти без заздалегідь вбудованого водяного знаку.

SVM – це алгоритм опорно-векторної машини (SVM), який широко використовується для класифікації аудіофайлів. За допомогою статистичного навчання класифікаторів, основна ідея алгоритму полягає в створенні гіперплощини, що розділяє об'єкти на класи найбільш оптимальним способом. SVM використовується для вирішення багатьох практичних завдань, таких як виявлення облич, розпізнавання тривимірних (3D) об'єктів і т.д. Проте для SVM можуть бути використані лише для задачі класифікації аудіофайлів та задачі ідентифікації голосу.

Слуховий момент Церніке – це один з методів, що дозволяє розпізнавати стиснену аудіоінформацію. На сьогодні стиснуті аудіо формати, такі як MP3, перетворилися на домінуючі формати збереження музики на персональних комп'ютерах та передачі її через Інтернет. Тому необхідно мати змогу розпізнавати безпосередньо стиснені аудіофайли без їх розпаковки, що, безумовно, буде більш ефективним.

Власне, функція "момент Церніке" використовується і методами обробки зображень, для розпізнавання зображень, створення водяного знаку зображення, розпізнавання людського обличчя, тощо. Моменти Церніке будуються набором комплексних поліномів, які утворюють повний ортогональний базис, визначений на площині $x^2 + y^2 \leq 1$.

Незручність безпосереднього застосування моменту Церніке для обробки аудіоінформації полягає в тому, що звук, по суті, є 1D-даними, тоді як моменти Церніке пристосовані тільки для 2D-даних. Отже, перед тим, як застосувати їх для обчислення моменту, необхідно привести аудіосигнали до 2D-форми.

Комп'ютерний зір – це спосіб, який пов'язаний з теорією та технологією побудови штучних систем, які отримують інформацію з зображень або багатовимірних даних.

Комп'ютерний зір застосовує машинне навчання для розпізнавання шаблонів, які забезпечують інтерпретацію зображень. Цей спосіб схожи1 на процес візуального мислення людського зору: виникає можливістю розрізняти об'єкти, класифікувати їх, сортувати їх за розмірами.

Комп'ютерний зір використовує зображення як вхідні дані і видає вихідні дані у вигляді інформації про розміри, інтенсивність кольору тощо. Наприклад, відеокамера, встановлена на автомобілі без водія, повинна виявляти людей на шляху автомобіля і відрізняти їх від транспортних засобів та інших об'єктів.

Кінцева мета полягає у використанні комп'ютерів для імітації людського зору, включаючи навчання і здатність виконувати обчислення на основі візуальних вхідних даних.

Комп'ютерний зір можна використати для ідентифікації аудіофайлів, якщо використовувати спектрограму кожного аудіозапису в якості 2D зображення.

Для даного рішення було використано метод акустичного відбитку, так як даний метод є більш чутливим до змін у аудіоінформації, завдяки використанню набору хеш-функцій. Також метод акустичного відбитку не потребує додавання метаданих або водяних знаків до аудіофайлу.

1.4. Формалізація постановки задачі дослідження

Метою даного дослідження є вивчення теоретичного підґрунтя систем ідентифікації аудіосигналів, а саме технологій декодування аудіофайлів, отримання частотних характеристик аудіосигналів, обчислення акустичних відбитків аудіосигналів, пошук готових рішень та програмних модулів, що допомають вирішенню проблеми ідентифікації аудіофайлів.

Задачею даного бакалаврського проекту є розробка веб-орієнтованої системи для визначення збігів в аудіофайлах. Система має забезпечити зручний для користувача інтерфейс, роботу в режимі on-line, бути стійкою до помилково негативних та помилково позитивних результатів.

2. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ПРОЕКТУ

2.1. Мова програмування JavaScript

JavaScript - це інтерпретована, об'єктно-орієнтована мова програмування, найбільш відома як мова сценаріїв для веб-сторінок [5]. Це прототипна, багатопарадигмова мова сценаріїв, з динамічною типізацією, що підтримує об'єктно-орієнтовані, імперативні та функціональні стилі програмування.

Сьогодні JavaScript може виконуватися не тільки в браузері, але і на сервері, або фактично на будь-якому пристрої, що має спеціальну програму під назвою JavaScript engine. Браузер має вбудований движок, який іноді називають «віртуальною машиною JavaScript».

Різні движки мають різні «кодові назви». Наприклад:

- Google V8, який використовується в браузері Google Chrome і останніх версіях браузера Opera. Його також використовує Node.js.
- SpiderMonkey - швидкий інтерпретатор, використовується в браузері Mozilla та в інших продуктах компанії Firefox.

Існують і інші кодові назви, як "Trident" і "Chakra" для різних версій IE, "ChakraCore" для Microsoft Edge, "Nitro" і "SquirrelFish" для Safari і т.д. [7].

Сучасна JavaScript - це «безпечна» мова програмування. Вона не забезпечує низькорівневий доступу до пам'яті або CPU, оскільки була створена для браузерів, які не потребують цього. Можливості JavaScript значною мірою залежать від середовища, у якому вона працює. Наприклад, Node.js підтримує функції, які дозволяють JavaScript читати та записувати файли, виконувати мережеві запити тощо. JavaScript у браузері забезпечує виконання маніпуляцій над веб-сторінками, взаємодією з користувачем і веб-сервером.

2.2 Бібліотека для створення інтерфейсів користувача React

React являє собою бібліотеку JavaScript з відкритим кодом, яка використовується для побудови інтерфейсів користувача. React була створена Йорданом Уолке, інженером-розробником програмного забезпечення Facebook. React була вперше застосована у Facebook у 2011 році та в Instagram у 2012 році.

В основі всіх React додатків лежать компоненти. Компонент - це самостійний модуль, який існує для відображення частини інтерфейсу. React надає можливість створювати елементи інтерфейсу, такі як кнопка або поле введення, як React компоненти. Компонент може включати в себе один або більше інших компонентів. У широкому розумінні, для написання React додатку створюються React-компоненти, які відповідають різним елементам інтерфейсу. Потім ці компоненти організовуються всередині компонентів вищого рівня, які визначають структуру програми.

На відміну від раніше розроблених бібліотек та фреймворків, React використовує не безпосередньо об'єктну модель документу браузера, а віртуальний DOM. Тобто, замість того, щоб маніпулювати DOM у браузері після зміни даних, React змінює віртуальний DOM. Після оновлення віртуального DOM, React визначає, які зміни потрібно внести до фактичного DOM браузера.

У React замість звичайного JavaScript для шаблонування використовується JSX. JSX - це проста JavaScript, який дозволяє цитувати HTML і використовує синтаксис HTML тегів для компонентів.

Поруч з React, для збирання модулів в розробленому додатку використовується Webpack. Webpack створює граф залежностей, що складається з різних модулів, які додаток використовує для функціонування. Коли Webpack обробляє програму, він починає зі списку модулів, визначених у командному рядку або у файлі налаштувань. Починаючи з точок входу, визначених у файлі налаштувань, Webpack рекурсивно будує графік залежностей, що включає в себе всі модулі, які

потребує додаток, а потім поєднує всі ці модулі в невелику кількість пакетів - для завантаження браузером. Завдяки широкому використанню React і Hot Module Replacement, Webpack почали використовувати в інших середовищах, таких як Ruby on Rails.

Для трансляції нової версії сиснтаксису JavaScript у старішу версію, яка підтримується усіма браузерами, було використано Babel. Це робить доступними всі нові інстремнти, які був додані до JavaScript з новою специфікацією ES6. З усіх трансляторів ES6, Babel володіє найбільшим рівнем сумісності з специфікацією ES6. Він дозволяє використовувати практично всі нові функції, які надає ES6 сьогодні, підтримуючи зворотну сумісність для старих браузерів.

2.3 Мова програмування Python

Python – це об'єктно-орієнтована мова програмування високого рівня з інтегрованою динамічною семантикою [9]. Вона надзвичайно приваблива в області швидкої розробки додатків, оскільки пропонує динамічну типізацію та можливості прив'язки динамічних назв до функцій та змінних.

Переважна більшість веб-додатків і платформ створені на Python, включаючи пошукову систему Google, YouTube і веб-орієнтовану систему транзакцій Нью-Йоркської фондової біржі. NASA фактично використовує Python для програмування обладнання та космічної техніки.

Python є інтерпретованою мовою, що означає, що програми, написані на Python, не потрібно заздалегідь компілювати для запуску, що полегшує тестування невеликих фрагментів коду та написання коду на Python для різних платформ. Вперше розроблений в кінці 80-х років Гвідо ван Россумом, в даний час найпоширенішою є третя версія Python, випущена в 2008 році.

Переважна більшість бібліотек, що використовуються для аналізу даних або машинного навчання, мають інтерфейси Python, що робить мову

найпопулярнішим командним інтерфейсом високого рівня для бібліотек машинного навчання та інших числових алгоритмів.

Стандартна бібліотека Python надає модулі для загальних завдань програмування - математика, обробка рядків, доступ до файлів і каталогів, мережа, асинхронні операції, потокова робота, багатопроцесорне керування тощо. Вона також включає в себе модулі, які керують завданнями програмування високого рівня, необхідними для сучасних додатків: читання і написання структурованих форматів файлів, таких як JSON і XML, маніпулювання стислими файлами, робота з Інтернет-протоколами і форматами даних (веб-сторінки, URL-адреси, електронна пошта). Більшість будь-якого зовнішнього коду, який відкриває інтерфейс C-сумісної зовнішньої функції, можна отримати за допомогою модуля cytpes Python.

Подібно до C #, Java та Go, Python має керований «збирач сміття», тобто програміст не повинен реалізовувати код для відстеження та знищення об'єктів, які невикористовуються додатком. Як правило, «збір сміття» відбувається автоматично у фоновому режимі, але якщо це створює проблеми з продуктивністю, його можна ініціювати вручну або повністю вимкнути.

Важливим аспектом мови є її динамічність. Python працює з функціями та модулями, як з об'єктами. Розробники можуть виконувати складні маніпуляції об'єктів лише з декількома інструкціями, і навіть розглядати частини програми як абстракції, які можуть бути змінені.

Одним з поширених недоліків Python, порівнюючи з іншими мовами програмування, є повільна обробка даних. Програми на Python зазвичай працюють набагато повільніше, ніж відповідні програми на C/C ++ або Java. Деякі програми на Python будуть повільнішими на порядок або більше. Це пов'язано з тим, що динамічні об’єкти в Python ускладнюють оптимізацію мови, навіть при компіляції.

2.4 Мікрофреймворк Flask

Flask – це веб-фреймворк Python [11]. Flask спочатку був створений Арміном Роначером, як жарт до Дня сміху в квітні 2010 року. Незважаючи на це, Flask став дуже популярним як альтернатива проектам Django з їхньою монолітною структурою та залежностями.

Одним з проектних рішень під Flask є те, що прості завдання повинні бути простими; вони не повинні займати багато коду.

Основна причина, чому Flask називається «мікрофреймворком» - це ідея зберегти ядро простим, але мати можливість його розширення додатковим функціоналом. У ньому немає абстрактного рівня бази даних, немає валідації форм або всього того, що вже є в інших бібліотеках. Однак, Flask підтримує розширення, які можуть додати необхідну функціональність і імплементує їх так, як ніби вони вже були вбудовані спочатку. На даний час вже є розширення: валідації форми, підтримка завантаження файлів, різні технології аутентифікації та ін.

Кожен додаток Flask є екземпляром flask.Flask класу. Об'єкт Flask реалізує програму і діє як центральний об'єкт. Клас flask.Flask відповідає за обробку всіх функцій перегляду, маршрутизації URL-адресів і налаштування шаблонів.

Отже для створення програмного продукту доцільно використовувати мову програмування JavaScript та бібліотеку React з її інфраструктурними компонентами, для розробки інтерфейсу користувача. Для створення серверної частини додатку можна використати мікрофреймворк Flask та мову програмування Python.

Такий підхід дасть можливість забезпечити:

- підтримку популярних браузерів;
- завантаження користувачем та обробки стиснених звукових файлів;
- маштабованість додатку.

3. СИСТЕМА ВИЗНАЧЕННЯ ЗБІГІВ В АУДІОФАЙЛАХ

3.1. Декодування аудіофайлів

Звук - це механічні коливання, які поширюються в твердих, рідких і газоподібних середовищах в формі пружних хвиль [15]. Коли хвиля досягає вуха, зокрема - барабанної перетинки, приводяться в рух слухові кісточки, які передають коливання далі, до волоскові клітини, розташованим у внутрішньому вусі. В результаті механічні коливання перетворюються в електричні імпульси, які передаються по слуховим нервах в мозок людини.

Пристрої для запису звуку досить точно імітують вищеописаний процес, конвертуючи тиск звукової хвилі в електричний сигнал. Звукова хвиля в повітрі - це безперервний сигнал, представлений областями стиснення і розрідження. Мікрофон, перший електронний компонент, з яким зустрічається звуковий сигнал, перетворює його в сигнал електричний, який все ще залишається безперервним. Подібні сигнали в цифровому світі не використовуються, тому, перед зберіганням і обробкою в цифрових системах, їх потрібно перетворити в дискретну форму. Робиться це за допомогою вибірки значень, що представляють значення амплітуди сигналу.

У процесі подібного перетворення проводиться квантування аналогового сигналу. Таким чином, перетворння сигналу вже не є одномоментним, аналого-цифровий перетворювач виконує безліч операцій по перетворенню дуже маленьких частин аналогового сигналу в цифровий. Цей процес називають дискретизацією або семплінгом.

Завдяки теоремі Котельникова відомо, яка частота дискретизації потрібна для того, щоб точно представити безперервний сигнал, обмежений певною частотою:

$$f_{дискр.} = 2 * F_{max}, \quad (3.1)$$

де $f_{дискр.}$ – частота дискретизації;

F_{max} – максимальна частота сигналу.

Зокрема, для того, щоб врахувати весь частотний спектр звуків, доступних людському слуху, необхідно використовувати частоту дискретизації, що вдвічі перевищує верхню межу частот, які чує людина. А саме, людина може чути звуки в діапазоні приблизно від 20 Гц до 20000 Гц. В результаті звук найчастіше записують з частотою дискретизації 44100 Гц. Саме ця частота дискретизації використовується в компакт-дисках. Вона ж найчастіше застосовується для кодування звуку в групі стандартів MPEG-1 (VCD, SVCD, MP3).

Широкому використанню частоти дискретизації в 44100 Гц посприяла, переважно, корпорація Sony. Свого часу звукові доріжки, закодовані таким способом, зручно було поєднувати з відео в стандартах PAL (25 кадрів в секунду) і NTSC (30 кадрів в секунду), працювати з ними, використовуючи існуюче обладнання. Дуже важливо і те, що ця частота достатня для якісної передачі звуку в діапазоні до 20000 Гц. Цифрове звукове обладнання, що використовує цю частоту дискретизації, цілком відповідало за якістю аналогового обладнання тих часів, коли відбувалося становлення стандартів цифрового звуку.

Обробка аудіофайлу починається з декодування аудіосигналу, який упакований в файл. MPEG кодування аудіофайлу під назвою MP3 стало одним з найпопулярніших стандартів для цифрового аудіо та відеотрансляції. Основна ідея кодування MP3 і звукового кодування в цілому - це видалення акустично неактуальної інформації із звукового сигналу, для зменшення його розміру.

MPEG є стисненням з втратами, тобто деяка аудіоінформація, звичайно, втрачається за допомогою цих методів стиснення. Цю втрату навряд чи можна помітити, оскільки метод стиснення намагається зробити її непомітною для людського слуху. Використовуючи декілька складних математичних алгоритмів, при кодуванні втрачаються лише ті складові звуку, які важко почути навіть у вихідній формі.

Стандарт кодування аудіосигналів MPEG-1 (описаний в ISO/IEC 11172-3) описує три шари кодування звуку з такими властивостями [16]:

- один або два аудіоканали;
- частота дискретизації 32 кГц, 44,1 кГц або 48 кГц;
- швидкість бітів від 32 кбіт/с до 448 кбіт/с.

Протягом останніх кількох років популярність MP3 зросла завдяки високим коефіцієнтам стиснення, які використовуються MP3-кодеками в різних автономних програвачах і ручних пристроях. Психоакустична модель, модифіковане дискретне косинусне перетворення (MDCT) і кодування Хаффмана відіграють важливу роль у досягненні високих коефіцієнтів стиснення. MP3 являє собою ланцюжок фреймів (блоків), в яких містяться закодовані дані про аудіо в форматі PCM.

Потік бітів всередині файлу MP3 (рис. 3.1) містить кадри з наступними частинами:

- заголовок;
- службова інформація;
- основні дані;
- допоміжні дані.

Заголовок	Службова інформація	Основні дані	Допоміжні дані

Рис. 3.1 Потік бітів всередині файлу MP3

Розмір заголовку завжди становить 32 біта або 4 байти, інформація в заголовку підтверджує автентичність MP3 файлу. Заголовок не завжди має бути на початку кадру. Тому кожен заголовок починається з синхронізу, щоб позначити його положення в MP3-файлі.

Розмір службової інформації може бути 17 байт, якщо вона поширюється єдиним каналом, або 32 байтами, якщо вона поширюється двоканально. Службова інформація завжди знаходиться після заголовку. Вона містить всю інформацію для декодування основних даних. Наприклад, інформацію про вибір масштабного коефіцієнта, інформацію таблиці Хаффмана для гранул. Деталі, щодо організації основних даних, можуть бути відомі лише після вилучення службової інформації.

Допоміжні дані можуть бути визначені користувачем, для його потреб, а точна кількість бітів не вказується явно. Допоміжні дані починаються після кодованих бітів Хаффмана. Відстань між кінцем кодованих бітів Хаффмана і місцем розташування в потоці бітів, де вказується головний покажчик основних даних наступного кадру, - це кількість допоміжних бітів.

Декодування MP3 файлів визначено в стандарті ISO. Кожен кадр складається з 1152 зразків і завжди має заголовок, приєднаний до кожного кадру, пов'язаного з файлом MP3. Заголовок та службова інформація для конкретного кадру необхідна, щоб декодування виконувалося правильно. Першим і найголовнішим в процесі декодування є синхронізація декодера з вхідним потоком бітів. Синхронізація - це процес знаходження позиції першого заголовка і наступних. Як тільки це буде зроблено, організація закодованих даних повністю відома і процедура декодування може виконуватися. Структурна схема на рис. 3.2 дає уявлення про процедуру.

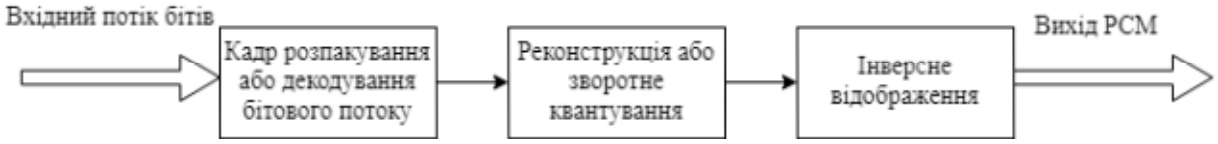

Рис. 3.2 Схема процедури декодування

Розпакування кадру являє собою процес знаходження заголовка бітового потоку, декодування службової інформації, коефіцієнтів масштабування декодування і декодування даних Хаффмана.

Кодування Хаффмана – це метод, який стискає і кодує заданий набір даних за допомогою кодової таблиці змінної довжини.

Основна ідея кодування Хаффмана проста. На вхід алгоритм кодування отримує деякі дані для стиснення, скажімо, список з 8-бітових символів. Потім створюється таблицю значень, в якій впорядковуються символи за частотою. Заздалегідь не відомо, як буде виглядати список символів, можна впорядкувати символи за ймовірністю їх виникнення в рядку. Потім призначаються кодові слова для таблиці значень, де короткі слова відповідатимуть найбільш ймовірним значенням. Кодове слово – це n-розрядне ціле число.

Символи, що зустічаються частіше, мають коротші коди. Ключовими питаннями щодо реалізації кодування Хаффмана є представлення кодової таблиці і відповідного алгоритму декодування.

Наприклад, в якості вхідних даних алгоритм отримав дуже довгий рядок з букв A, C, G і T. Можна помітити, що марно зберігати цей рядок, як рядок 8-бітних символів, тому такий рядок зберігається як рядок 2-бітних символів. Кодування Хаффмана може стискати рядок далі, якщо деякі з букв зустрічаються частіше, ніж інші. У даному прикладі заздалегідь відомо що "A" зустрічається у рядку з ймовірністю приблизно 0,4. Можна створити таблицю ймовірностей.

Декодування – це процес зворотний кодуванню. Якщо в якості вхідних даних є бітовий рядок, скажімо 00011111010, алгоритм читає біти, поки не знайдеться відповідність у таблиці. Рядок для прикладу вище декодується до AAATGC. Таблиця кодових слів має бути розроблена таким чином, щоб не було конфліктів кодових слів, наприклад, якщо символу A в таблиці відповідає код 0, а символу C

– код 01, і зустрічається в закодованому рядку 0, неможливо зрозуміти, який це символ.

Таблиця 3.1

Таблиця ймовірностей

Символ	Ймовірність зустріти символ у рядку
A	0,4
C	0,35
G	0,2
T	0,05

Таблиця 3.2

Таблиця з додаванням кодових слів

Символ	Кодове слово
A	0
C	10
G	110
T	111

Стандартний спосіб декодування Хаффман-кодованого рядка полягає в проходженні бінарного дерева, створеного з таблиці кодових слів. Коли зустрічається біт 0, алгоритм проходиться ліворуч по дереву, і прямо, коли зустрічається біт 1. Це найпростіший метод, який використовується в даному декодері.

Існує більш ефективний метод декодування рядка. Замість того, щоб ходити по дереву, можна по-іншому використати таблицю пошуку.

У таблиці 3.3 xx означає всі перестановки 2 біт - всі бітові шаблони від 00 до 11. Таблиця заповнена таким чином, що містить всі індекси від 000 до 111. Для декодування рядка за допомогою цієї таблиці аналізуються 3 біти в кодованому бітовому рядку. Бітовий рядок - 00011111010, тому індекс - 000. Це відповідає парі (A, 1), що означає, що знайдено значення A, і необхідно відкинути 1 біт з входу. Аналізуються ще 3 біти в рядку і процес аналізу повторюється.

Таблиця 3.3

Таблиця пошуку

Кодове слово	Символ
0xx	(A, 1)
10x	(C, 2)
110	(G, 3)
111	(T, 3)

Щоб зрозуміти, як кодування Хаффмана використовується MP3, необхідно зрозуміти, що саме кодується або декодується. Стислі дані, які необхідно розпакувати, є зразками частотної області. Кожен логічний кадр має до чотирьох блоків - по два на канал - кожен з яких містить до 576 частотних зразків. Для 44100 Гц звукового сигналу перша частотна вибірка (індекс 0) представляє частоти близько 0 Гц, тоді як остання вибірка (індекс 575) представляє частоту близько 22050 Гц.

Ці зразки поділяються на п'ять різних областей змінної довжини. Перші три області відомі як області великих значень, четверта область відома як область чотирьохрегіональна область, а п'ята область відома як нульова область. Зразки в нульовій області є нульовими, тому вони фактично не кодуються Хаффманом. Якщо регіони великих значень і чотирьохрегіональна декодуються до 400 вибірок, до решти 176 просто додається 0.

Три області великих значень представляють важливі нижчі частоти в аудіосигналі. Їх назва відноситься до інформаційного змісту: коли виконується декодування, ці області будуть містити цілі числа в діапазоні від мінус 8206 до 8206.

Ці три області великих значень кодуються трьома різними таблицями Хаффмана, визначеними у стандарті MP3. Стандарт визначає 15 великих таблиць для цих областей, де кожна таблиця виводить дві частотні вибірки для даного кодового слова. Таблиці призначені для максимального стиснення «типового» змісту частотних областей.

Для подальшого збільшення стиснення 15 таблиць поєднуються з іншим параметром, створюючи 29 різних способів, яким кожен з трьох областей може бути стиснутий. Службові дані містять інформацію про те, який з 29 способів був використаний.

Отже, було зроблено один з ключових кроків декодування MP3: декодування даних Хаффмана. Тепер необхідно зробити другий ключовий крок - повторне квантування.

Квантування - це апроксимація значень з великим діапазоном, значеннями з меншим діапазоном, тобто з використанням меншої кількості бітів. Наприклад, якщо приймається аналоговий аудіосигнал і відбирається на дискретних інтервалах часу, в результаті створюється дискретний сигнал - список зразків аудіосигналу. Оскільки аналоговий сигнал є безперервним, ці зразки будуть реальними значеннями. Якщо квантувати зразки аудіосигналу, скажімо, кожен реальний зразок цілим числом від мінус 32767 до плюс 32767, ми отримуємо цифровий сигнал - дискретний в обох вимірах.

Квантування може використовуватися як форма стиснення з втратами. Для 16-бітового PCM кожен зразок аудіосигналу може приймати одне з 216 значень. Якщо замість кожного зразка аудіосигналу наблизитися до діапазону від мінус 16383 до

плюс 16383, втрачається інформація, але заощаджується 1 біт даних на зразок аудіосигналу. Різниця між вихідним значенням і квантованим значенням відома як помилка квантування, яка призводить до виникнення шуму. Різниця між реальним зразком і 16-бітовою вибіркою настільки мала, що вона є незначною для більшості цілей, але якщо вилучити занадто багато інформації з вибірки, різниця між оригіналом та вибіркою буде відчутною для людського слуху.

Всі безперервні сигнали можна створити, за допомогою додавання синусоїд. Це означає, що якщо використовується чиста синусоїда, скажімо, при 440 Гц, яка квантується, то помилка квантування проявиться як нові частотні компоненти в сигналі. Квантована синусоїда насправді не є чистою синусоїдою. Ці нові частоти будуть по всьому спектру, вони і створюватимуть шум. Якщо помилка квантування мала, то величина шуму буде невеликою.

Людський слух не є ідеальним: якщо в критичній смузі є сильний сигнал, то шум, викликаний помилками квантування, буде замаскований.

Методи квантування можна записати як математичні вирази. Скажімо, існує реальний зразок в діапазоні від мінус 1 до плюс 1. Щоб квантувати це значення у формі, придатній для 16-бітового файлу WAV, потрібно помножити вибірку на 32727 і відкинути частку: q = floor (s * 32727). Повторне квантування в цьому простому випадку є поділом, де різниця між повторно квантованим зразком і оригіналом є помилкою квантування.

3.2. Дискретне перетворення Фур'є

Потрібно знайти спосіб отримання частотних характеристик сигналів, розгорнутих у часі.

У 19 столітті Жан Батист Джозеф Фур'є зробив видатне відкриття [17]. Полягає воно в тому, що будь-який сигнал у часовій області еквівалентний сумі деякої кількості простих синусоїдальних сигналів, за умови, що кожна синусоїда має

певну частоту, амплітуду і фазу. Набір синусоїд, які формують вихідний сигнал, називають рядом Фур'є.

Іншими словами, можна уявити практично будь-який сигнал, розгорнутий у часі, просто задавши набір частот, амплітуд і фаз, відповідних кожній з синусоїд, які цей сигнал формують. Таке уявлення сигналів називають набором частотних інтервалів. В якомусь сенсі, відомості про частотних інтервалах є чимось на зразок «відбитків пальців» або сигнатур сигналів, розгорнутих у часі, що є зрізом динамічних даних.

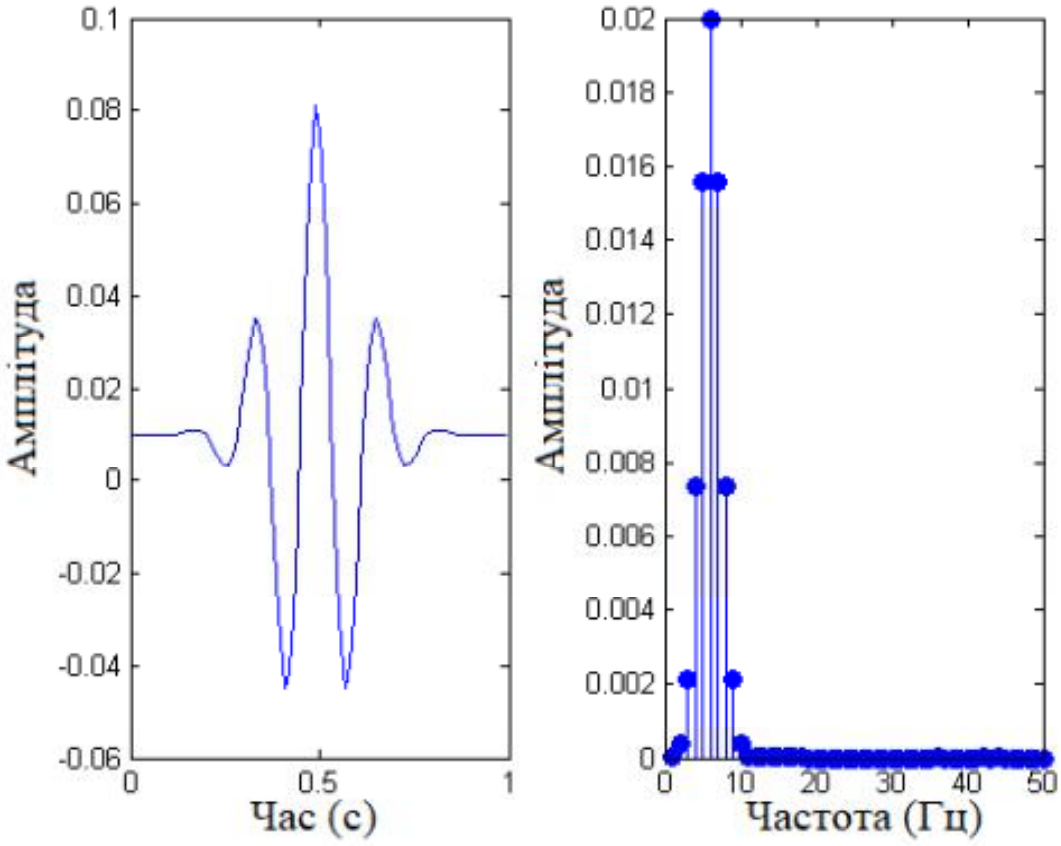

Рис. 3.3 Сигнал, розгорнутий в часі, і його частотні характеристики

Аналіз частотних характеристик сигналів значно полегшує вирішення безлічі завдань, пов'язаних з перетворенням сигналів. Оперувати такими характеристиками в сфері обробки цифрових сигналів, дуже зручно. Вони дозволяють вивчати спектр сигналу (його частотні характеристики), визначати, які

частоти в цьому сигналі є, а які - відсутні. Після цього можна зробити фільтрацію, посилити чи послабити деякі частоти, або просто розпізнати звук певної висоти серед наявного набору частот.

Для цього використовується дискретне перетворення Фур'є (ДПФ). ДПФ - це математичний метод аналізу Фур'є для дискретних сигналів. З його допомогою можна перетворити кінцевий набір зразків сигналу, взятих з рівними проміжками часу, в список коефіцієнтів кінцевої комбінації комплексних синусоїд, упорядкованих за частотою, кожна з яких має власну амплітуду і фазу, беручи до уваги, що ці синусоїди були дисретизовані з однією і тією ж частотою.

Один з найпопулярніших чисельних алгоритмів для обчислення ДПФ називається швидке перетворення Фур'є (ШПФ). Насправді, ШПФ представлене цілим набором алгоритмів. Серед них найчастіше використовуються варіанти алгоритму Кулі-Тьюки (Cooley-Tukey).

В основі цього алгоритму лежить принцип «розділяй і володарюй». В ході обчислень використовується рекурсивне розкладання вихідного ШПФ на дрібні частини. Пряме обчислення ШПФ для деякого набору даних n вимагає $O(n^2)$ операцій, а використання алгоритму Кулі-Тьюки дозволяє вирішити ту ж задачу з O(n*log(n)) операцій.

Ось кілька бібліотек на різних мовах програмування, що реалізують ШПФ:

- C - FFTW
- C ++ - EigenFFT
- Java - JTransform
- Python - NumPy
- Ruby - Ruby-FFTW3 (інтерфейс до FFTW)

Один з неприємних побічних ефектів ШПФ полягає в тому, що після проведення аналізу, втрачається інформація про час. Хоча, теоретично, подібного можна уникнути, але на практиці для цього знадобиться величезна обчислювальна

потужність. Наприклад, для трихвилинної пісні можна отримати звукові частоти і їх амплітуди, але ось де саме в творі ці частоти зустрічаються, не відомо. А це - найважливіша характеристика, яка робить музичний твір тим, чим він є. Потрібно дізнатися точні значення часу, коли з'являється кожна з частот.

Саме тому необхідно користуватися чимось на зразок «ковзаючого вікна» і піддавати трансформації лише ту частину сигналу, яка в це «вікно» потрапляє. Розмір кожного блоку можна визначити з використанням різних підходів. Наприклад, якщо записується двоканальний звук з розміром зразка аудіосигналу рівним 16 біт і з частотою дискретизації 44100 Гц, одна секунда такого звуку займе 176 Кб пам'яті (44100 зразків * 2 байта * 2 канали). Якщо встановити розмір ковзаючого вікна, рівний 4 Кб, то кожну секунду потрібно буде відстежувати 44 блоки даних.

У внутрішньому циклі обчислюються дані з тимчасової області (зразки звуку) у вигляді комплексних чисел з уявною частиною рівною 0. У зовнішньому циклі алгоритм ітерується по всіх блоках даних і для кожного з них запускає ШПФ-аналіз.

Результатом роботи алгоритму є набір комплексних чисел, які, будучи перенесеними на площину, називаються спектрограмою. Спектрограма - це візуальне представлення всіх трьох акустичних вимірювань: часу, частоти і амплітуди сигналу. Вона показує значення амплітуди для певного значення частоти в певний момент часу. По осі X відраховується час, вісь Y представляє частоту, а значення амплітуди позначається інтенсивністю кольору пікселя (рис. 3.4.).

Це досить докладний «портрет» аудіозапису, з якого можна (з певною апроксимацією) відновити вихідний трек. З точки зору ресурсів, зберігати такий «портрет» повністю недоцільно. Можна вибрати ключові точки на спектрограмі,

грунтуючись на інтенсивності спектра, щоб зберігати тільки найхарактерніші для цього треку значення, що зменшить обсяг збережених даних.

Головна складність тут - вибрати з величезної кількості частот саме найважливіші. Чисто інтуїтивно вибираються частоти з максимальними амплітудами (зазвичай їх називають піками).

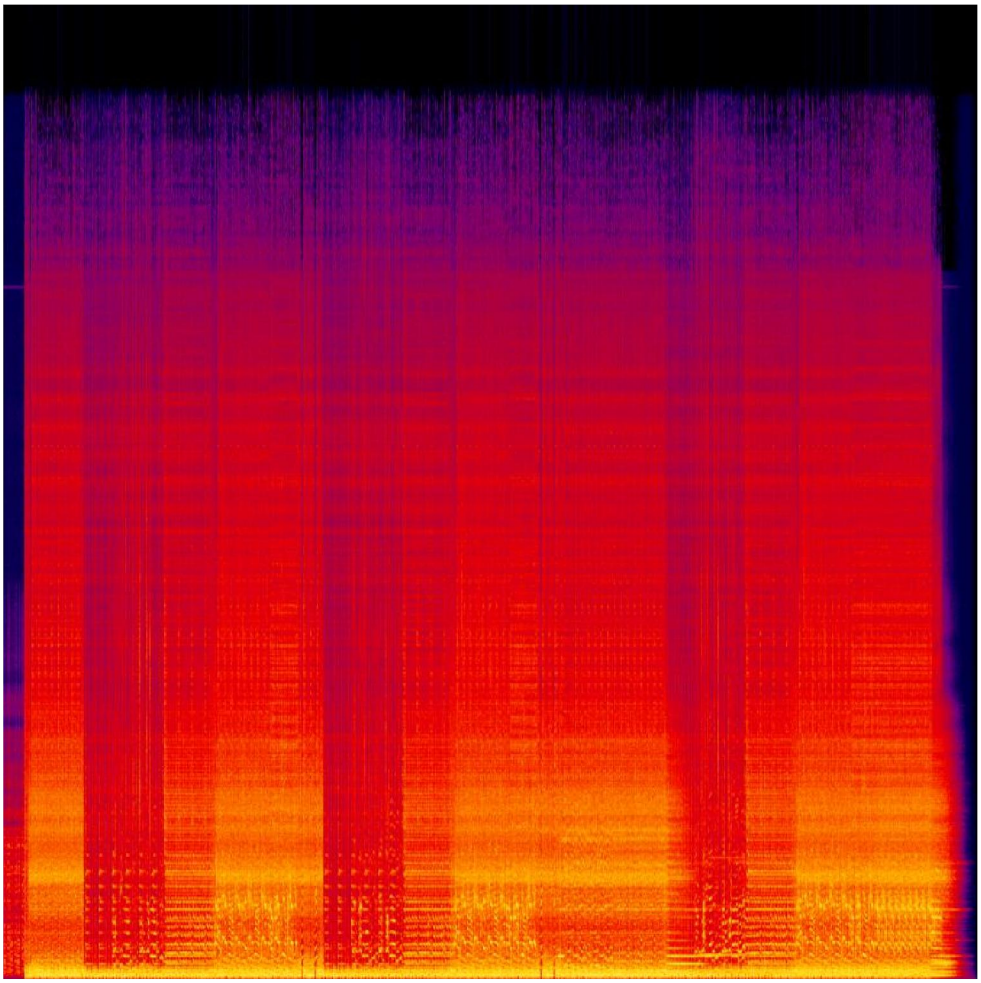

Рисунок 3.4 – Спектрограма

Однак, в одному музичному творі діапазон «сильних» частот може варіюватися, скажімо, від ноти «до» контроктави (32,70 Гц), до ноти «до» п'ятої октави (4186,01 Гц). Це - величезний інтервал. Тому, замість того, щоб відразу проаналізувати весь частотний діапазон, можна вибрати кілька дрібніших інтервалів. Вибирати потрібно грунтуючись на частотах, які зазвичай притаманні важливим музичним компонентам, і проаналізувати їх окремо. Наприклад, можна

скористатися інтервалами 30 Гц - 40 Гц, 40 Гц - 80 Гц і 80 Гц - 120 Гц для низьких звуків (сюди відноситься, наприклад, бас-гітара). Для середніх і вищих звуків застосовуються частоти 120 Гц - 180 Гц і 180 Гц - 300 Гц (сюди входить вокал і більшість інших інструментів).

Після того як, інтервали визначено, в них знаходять частоти з найвищими рівнями. Ці відомості і формують сигнатуру для конкретного аналізованого блоку даних, а вона, в свою чергу, є частиною сигнатури всього музичного твору.

Необхідно зібрати отримані результати в форму, зручну для подальшого аналізу. Кожен пік однозначно визначається двома числами - значеннями частоти і часу. Додавши всі піки для треку в один масив, отримаємо шуканий акустичний відбиток.

3.3. Отримання акустичного відбитку сигналу

Одна з перших систем акустичних відбитків була розроблений в 1990-х роках Е. Волдом та названа «Muscle Fish». Ця система була використана для ідентифікації коротких звуків, як дзвінок або оплески. Алгоритм дозволяв ідентифікувати аудіоінформацію, але не міг бути використаний для розпізнавання музичного треку на основі частини доріжки.

З 2000 року було випущено декілька систем акустичних відбитків, що дозволяли визначати музичні треки на основі їхнього змісту. Прикладами є система Relatable TRM, яка була ліцензована компанією Napster для фільтрації матеріалів, захищених авторським правом, і системи акустичних відбитків фірми Philips Research.

У 2005 році ця технологія Philips Research була ліцензована компанією Gracenote, яка реалізувала її у своїх продуктах розпізнавання звуку. Компанія Audible Magic, розробила свій алгоритм та використовують його для роботи у сфері розпізнавання ЗМІ та управління авторськими правами. Веб-сайт

MusicBrainz.org, який дозволяє користувачам автоматично керувати своїми музичними бібліотеками, спочатку використовувала Relatable TRM для ідентифікації аудіо, але коли було досягнуто меж масштабованості цього алгоритму, вони перейшли на систему відбитків звуку від компанії MusicIP.

Функція F відображає аудіооб'єкт X, що складається з великої кількості бітів, в акустичний відбиток, що складається з обмеженого числа бітів. Весь механізм обчислення акустичного відбитку аудіофайлу можна розглядати як хеш-функцію H, яка відображає об'єкт X у хеш. Головною перевагою хеш-функцій є те, що вони дозволяють порівняти два великих об'єкта X, Y, просто порівнюючи відповідні значення хешів H (X), H (Y). Рівність між H (X) та H (Y)означає рівність між X, Y, з дуже низькою ймовірністю помилки.

Отже, алгоритми акустичних відбитків перетворюють функції в числові коди або хеші, що представляють значення функції. Більшість алгоритмів акустичних відбитків обчислюють функції з аудіодомену, зазвичай використовуючи перетворення Фур'є.

Процес створення акустичного відбитку умовно можна поділити на такі етапи:

1. Попередня обробка

Попередня обробка включає перетворення аудіоданих у стандартний формат. Аудіосигнал оцифровується (при необхідності) і перетворюється в монопотік. Крім того, аудіосигнал часто змінюється на конкретну частоту дискретизації. Це може бути стандартна частота CD-дисків 44,1 кГц, але часто звук знижується до частоти від 4 кГц до 8 кГц. Це призведе до втрати частот вище 2 кГц до 4 кГц (частоти Найквіста), але ці більш високі частоти містять менш важливу інформацію для розпізнавання звуку, це різко покращує ефективність алгоритму перетворення. Нарешті, може бути застосована нормалізація, яка забезпечить знаходження амплітуди аудіоданих у стандартному діапазоні для кадрування і перекриття. При цьому вибірка зменшує високу частоту інформації;

високочастотна складова зазвичай містить менше енергії і тому більш чутлива до спотворень і менш стабільна.

2. Перекриття

Акусичні відбитки потрібно обчислювати для невеликих уривків твору, а також для повного треку, вхідні дані розбиваються на невеликі кадри тривалістю кілька мілісекунд. Завдяки такому кадруванню враховується лише короткий фрагмент твору. Таким чином, сигнал залишається рівним на довжині кадру. Тому можна визначити особливості сигналу всередині кадру. Щоб уникнути розривів на початку і в кінці кадру, застосовується функція вікна. Функції вікна є функціями Хеммінга і фон Ханна.

3. Трансформація

Третій етап, який є однаковим для більшості систем акустичних відбитків – це етап трансформації. Надлишковість даних на даному етапі значно зменшується. Більшість систем застосовують реалізацію швидкого перетворення Фур'є CooleyTukey для полегшення ефективного стиснення, видалення шуму та подальшої обробки. Були запропоновані деякі інші перетворення - дискретне косинусне перетворення, трансформація Хаара або трансформація Уолша-Адамара. В розробленому додатку був застосований алгоритм швидкого перетворення Фур'є.

4. Вилучення ознак

Вилучення ознак, в основному, спрямоване на зменшення розміру у вигляді ефективного опису основного сигналу. Крім того, за допомогою функцій, що базуються на найбільш надійних елементах сигналу, може бути підвищена надійність до спотворень.

5. Подальша обробка

Цей етап може бути використаний для нормалізації ознак, щоб представити дані в ефективній формі. Порядок цих кроків може бути різним, повторюватися або застосовуватися в різних масштабах часу або частоти.

На закінчення можна сказати, що кожен із згаданих вище етапів спрямований на одну або декілька з наступних задач:

- зменшення розміру даних та їх компактне представлення;
- підвищення стійкості до спотворень;
- підкреслення унікальних характеристик сигналу.

Розглянемо три категорії подання акустичних відбитків:

1. Відбиток фіксованого розміру

Розмір відбитка аудіосигналу не залежить від розміру аудіофайлу. Музика нестаціонарна, частини з різними сигнальними і статистичними характеристиками змішуються в кінцевому поданні.

Існує три недоліки такої системи. По-перше, коли різні частини змішуються разом в одній моделі, дискримінаційні характеристики таких фрагментів втрачаються в процедурі моделювання; при ідентифікації більш коротких фрагментів існує лише часткове зпівставлення з моделлю, отриманою для всієї системи. По-друге, інформація про синхронізацію і тимчасовий порядок ознак є відмінною особливістю сигналу. По-третє, відмінності відбитків аудіосигналу не можна використовувати для визначення відмінностей між сигналами.

Однією з переваг втрати тимчасової інформації є те, що модель потенційно стає незалежною від спотворень часу масштабування.

2. Постійна швидкість

Більшість систем акустичних відбитків обчислюють значення хеш-функцій на регулярних часових інтервалах (кадрах). Таким чином, розмір відбитків є пропорційним розміру аудіофайлу. Основною перевагою є те, що характеристики сигналу, які змінюються з часом, не змішуються в кінцевому відбитку. Крім того,

може бути гарантована кількість інформації, отриманої за певний часовий проміжок. Нарешті, коли порівнюють відбиток сканованої версії з відбитком первинного неспотвореного запису, відмінність відбитків можна використовувати для локалізації змін у спотвореній версії.

3. Акустичні відбитки змінної швидкості

Для ефективного подання частота відбитків змінюється в залежності від акустичних подій. Таким чином, відбиток лише відображає характеристики базового акустичного сигналу. Наприклад в алгоритмі Shazam, спектральні точки піку, які є найбільш значними як в частоті, так і в часовому вимірі, являють собою акустичний відбиток. Це призводить до компактності акустичних відбитків. Проте, не можна гарантувати обсяг інформації, отриманої в певному часовому відрізку. Частина відбитка, яка відповідає певному моменту часу, називається суб-відбитком аудіосигналу. Таким чином, відбиток аудіосигналу задається послідовністю суб-відбитків аудіосигналу. Кілька суб-відбитків аудіосигналу, що використовуються для ідентифікації, називаються блоком відбитків аудіосигналу.

Розглянемо існуючі технології створення акустичних відбитків:

1. Echoprint

Технологія Echoprint використовується для будь-якої сервісної системи або системи ідентифікації з великим набором аудіоданих. Echoprint - це технологія ідентифікації музики з відкритим вихідним кодом. Echoprint є ефективною і має високу продуктивність, генеруює десятки хешів в секунду з вхідного аудіо (мікрофон або файли), а потім зіставляє ці хеші в інвертованому індексі для запитів. Алгоритм Echoprint знаходить моменти часу, коли відтворюються певні музичні ноти. Відповідні записи визначають, знаходячи ідентичні хеші в базі даних. Алгоритм технології Echoprint:

- аудіосигнал перетворюється в моно сигнал, а частота вибірки зменшується до 11025 Гц;

- з вхідного сигналу генерується 40-полюсний лінійний фільтр-предикатор для виконання «відбілювання» сигналу;

- після того, як аудіосигнал був відібраний і «відбілений», він перетворюється в частотну область. Для виконання перетворення використовується банк косинусних фільтрів з діапазоном 128 смуг;

- банк фільтрів переміщується по сигналу, розбитому на 32 зразка;

- результуючі смуги частот згруповуються у вісім однаково розташованих комірок шляхом підсумовування абсолютної різниці сусідніх смуг. Вісім комірок рівномірно розподілені від 0 Гц до 5512.5 Гц;

- хешовані «ехопринти» обчислюються на основі різниці часу між музичними відрізками в кожній смузі;

- у кожному діапазоні для вимірювання амплітуди смуги використовується відстеження амплітуди. Коли амплітуда досягає порогу, початок реєструється;

- два значення хешу та індексу смуги зберігаються в 40 біт (5 байтів) (два байти для кожної дельти і 1 байт для індексу смуги). Число зменшується до 32 біт з алгоритмом Мурмур Хаш;

- кожен набір даних генерує шість хешів;

- хеші поєднуються за часом, коли відбувається початок в аудіофайлі. І зберігаються в базі даних.

2. AcoustID (Chromaprint)

Це технологія з відкритим вихідним кодом і останнім часом найчастіше використовувана технологія акустичних відбитків. Вона була створена Лукасом Лалінським і оприлюднена в січні 2011 року. Chromaprint - основний компонент проекту AcoustID. Це клієнтська бібліотека, яка реалізує власний алгоритм вилучення акустичних відбитків з будь-якого джерела звуку. Використовуючи техніку AdaBoost, алгоритм генерує 16 фільтрів.

Ці 16 фільтрів попередньо обчислюються частиною алгоритму Chromaprint і не змінюються. До кожного кадру аудіосигналу застосовуються 16 створених фільтрів. Для застосуватння фільтру, підсумовується кількість енергії в білій області і віднімається кількість енергії в чорній області, що призводить до одиничного значення. Кожен з фільтрів квантує енергетичне значення до 2-бітного числа. 2-бітове значення кодується за допомогою кодування Грея, і записується як двійкова послідовність, де кожне значення відрізняється від попереднього і наступного значення лише одним бітом. 2-бітові хеш-значення кожного з 16 фільтрів перетворюються в єдине 32-розрядне ціле число, що представляє суб-відбиток аудіосигналу.

Суб-відбитки зберігаються в таблиці з інвертованими індексами, що вказують на запис, в якому вони знаходяться, повний акустичний відбиток аудіосигналу зберігається в базі даних.

3. Panako

Panako - технологія акустичних відбитків. Деякі частини Panako були натхненні реалізацією Дена Елліса. Алгоритм технології Panako використовує піки в амплітуді спектру в кожному кадрі аудіосигналу. Технологія формує відбитки аудіопотоку і зберігає ці відбитки в базі даних.

Алгоритм технології Panako, дозволяє надійно і швидко ідентифікувати аудіозаписи, навіть якщо вони були прискорені, розтягнуті в часі або зміщені відносно опорного звуку. Аналогові фізичні носії, такі як воскові циліндри, дротові записи, магнітні стрічки і грамофонні записи, можуть бути оцифровані при неправильній або різній швидкості відтворення. Навіть коли калібровані механічні пристрої використовуються в процесі оцифрування, аудіофайли вже могли бути записані з небажаною швидкістю.

Щоб ідентифікувати дублікати в оцифрованому архіві, алгоритм пошуку повинен компенсувати зміни швидкості відтворення. Поруч із випадковими

змінами швидкості іноді під час радіотрансляцій вводяться навмисні маніпуляції зі швидкістю: іноді пісні грають трохи швидше, щоб вписатися в часовий інтервал. Під час набору DJ-дисків майже завжди відбуваються зміни швидкості. Щоб правильно ідентифікувати аудіофайл в цих випадках, також потрібний алгоритм пошуку музики, надійний у відношенні перенесення тону, часу розтягування та зміни швидкості. Алгоритм технології Panako дозволяє відстежити такі зміни при збереженні масштабованості та надійності.

Алгоритм технології Panako:

- локальні максимуми обчислюються для спектрограми запиту. Вони поєднуються, щоб утворити відбиток аудіосигналу;
- для кожного відбитку аудіосигналу обчислюється відповідне значення хешу;
- набір хешів порівнюється з хешами, що зберігаються в базі даних, кожен точний збіг повертається;
- збіги зі значенням аудіоідентифікатора, меншим, ніж певний поріг, видаляються, фактично відкидаючи випадкові збіги;
- залишкові збіги перевіряються вирівнюванням, як за частотою, так і за часом.

Порівняємо дані технології, використовуючи різні за часом частини аудіозапису (Табл. 3.4).

Білий шум - це сигнал, що має однакову інтенсивність на різних частотах, що надає йому постійну спектральну потужність. Білий шум є дискретним сигналом, зразки якого розглядаються як послідовність послідовно некорельованих випадкових величин з нульовою середньою і кінцевою дисперсією, іншими словами, незалежні та ідентично розподілені випадкові величини є найпростішим зображенням білого шуму. Сигнал називають "білим шумом", якщо він має

плоский спектр на відповідному діапазоні частот. Для звукового сигналу відповідним діапазоном є смуга звукових частот від 20 до 20 000 Гц.

Таблиця 3.4

Точність ідентифікації аудіосигналів (для 300 аудіосигналів)

Технології	0-5 секунд аудіосигналу	0-15 секунд аудіосигналу	0-30 секунд аудіосигналу	Середня точність ідентифікації аудіозаписів
Echoprint	45%	95%	95.67%	78.56%
Chromaprint	0.33%	96.67%	99.67%	65.56%
Panako	60.67%	95%	100%	85.22%

Порівняємо дані технології, використовуючи аудіозапис, з доданим білим шумом (Табл. 3.5).

Таблиця 3.5

Точність ідентифікації аудіосигналів з доданим білим шумом (для 300 аудіосигналів)

Технології	0-15 секунд аудіосигналу	0-30 секунд аудіосигналу	Середня точність ідентифікації аудіозаписів
Echoprint	66.69%	80.19%	73.44%
Chromaprint	73.59%	77.55%	75.57%
Panako	93.06%	97.68%	95.37%

Порівняємо дані технології, використовуючи аудіозапис, з доданим рожевим шумом (Табл. 3.6).

Таблиця 3.6

Точність ідентифікації аудіосигналів з доданим рожевим шумом

(для 300 аудіосигналів)

Технології	0-15 секунд аудіосигналу	0-30 секунд аудіосигналу	Середня точність ідентифікації аудіозаписів
Echoprint	58.67%	73.67%	66.17%
Chromaprint	40%	80.67%	60.335%
Panako	77%	92.33%	84.665%

Рожевий шум є сигналом з таким частотним спектром, що спектральна потужності (потужність сигналу на один частотний інтервал) обернено пропорційна частоті сигналу. У рожевому шумі кожна октава (зменшення вдвічі або подвоєння частоти) має однакову кількість шумової енергії.

Рожевий шум є найбільш поширеним сигналом у біологічних системах, тому аудіосистемам необхідно забеезпечувати фільтрацію рожевого шуму.

Отже, на основі аналізу існуючих технологій генерації акустичного відбитку, можна зробити висновок, що основними вимогами до алгоритму генерації акустичного відбитку є:

1. Надійність до помилково негативних результатів

Помилково негативний результат - алгоритм не дає результату, враховуючи наявний результат. Для того, щоб досягти високої надійності, відбиток повинен

бути вилучений на основі перцептивних ознак, які не змінюються в залежності від погіршення сигналу.

2. Надійність до помилково позитивних результатів

Помилково позитивний результат - алгоритм повертає результат, але він неправильний. Це важливий фактор, оскільки помилкові спрацьовування можуть викликати проблеми з авторським правом, розподілом роялті, користувачами, які завантажують пісні.

Технології Echoprint, Chromaprint і Panako є надійними, оскільки коефіцієнт повернення неправильного результату у всіх алгоритмів незначний.

3. Розмір відбитків

Ця характеристика показує, скільки місця потрібно для зберігання кожного відбитка. Це може бути важливим фактором для систем, де відбитки для ідентифікації відправляються за допомогою мереж даних стільникового зв'язку. Panako зберігає значення хешу, а також відображає відбиток графічним методом, тому для зберігання таких відбитків потрібно більше місця.

Повернемося до вихідної задачі - порівняти ці треки за допомогою відбитків і з'ясувати, схожі (однакові) вони чи ні. Кожен відбиток - масив значень. Можна порівнювати їх поелементно, зміщуючи треки по часовій шкалі відносно один одного (зміщеннння потрібне, наприклад, щоб врахувати тишу на початку або в кінці треку). На одних зміщеннях збігів в відбитках буде більше, на інших - менше.

Можна використати переваги структури самого відбитка. Відбиток - це масив, тому потрібно розглядати окремі його елементи, які відповідають пікам спектру, розбиваючи кожен відбиток на хеші.

3.4. Пошук найбільшої загальної підпослідовності

Для слухача невелике прискорення аудіофайлу не сприймається як істотна відмінність. На жаль, алгоритм порівняння відбитків вважатиме такі треки зовсім різними.

Щоб це виправити, було додано пошук найбільшої загальної підпослідовності (Longest Common Subsequence) в двох відбитках. Адже амплітуда і частота не змінюються, змінюється в цьому випадку тільки відповідне значення часу, а загальний порядок проходження точок зберігається.

Знаходження LCS дозволяє визначити коефіцієнт «стиснення» або «розтягування» сигналу по шкалі часу. До одного з акустичних відбитків застосовується знайдений коефіцієнт, далі відбитки аудіосигналу порівнюються як зазвичай.

Проблема пошуку найбільшої загальної підпослідовності полягає в наступному. Наведено два рядки: рядок S довжини k і рядок T довжини d. Метою є створення найдовшої спільної підпослідовності: найдовша послідовність символів, яка з'являється зліва направо (але не обов'язково в суміжному блоці) в обох рядках.

Наприклад, є два рядки: S = ABAZDC і T = BACBAD. У цьому випадку LCS має довжину 4 і є рядком ABAD. Потрібно знайти відповідності між деякими з букв в S і деякими з букв в T. Можна вирішити проблему LCS за допомогою динамічного програмування. Для спрощення алгоритму, спочатку потрібно визначити довжину LCS, тоді можна змінити алгоритм, щоб створити власну послідовність. LCS [i, j] - довжина LCS для S[1..i] та T [1..j].

Випадок 1: якщо S [i] ≠ T [j]. Тоді бажана підпослідовність повинна ігнорувати або S[i], або T[j], тому:

$$LCS [i, j] = \max (LCS [i - 1, j], LCS [i, j - 1]). \quad (3.2)$$

Випадок 2: якщо S [i] = T [j]. Тоді LCS з S [1..i] і T [1..j] можуть також збігатися з ними. Наприклад, якщо загальна підпослідовність, що є відповідністю S [i] і якось розташування в T, наприклад, T [j]. Отже, у цьому випадку:

$$LCS [i, j] = 1 + LCS [i - 1, j - 1]. \quad (3.3)$$

Тому, можна зробити два цикли (за значеннями i та j), заповнивши LCS. Ось як виглядає матриця для прикладу наведеного вище, де S вздовж лівого стовпця, а T - вздовж верхнього рядку (Табл. 3.7).

Таблиця 3.7

Матриця алгоритму LCS

	B	A	C	B	A	D
A	0	1	1	1	1	1
B	1	1	1	2	2	2
A	1	2	2	2	3	3
Z	1	2	2	2	3	3
D	1	2	2	2	3	4
C	1	2	3	3	3	4

Ця матриця заповнюється рядком за рядком, тому це займає O(k*d). Шуканий результат (довжина LCS S і T) знаходиться в нижньому правому куті. Щоб знайти послідовність, необхідно рухатися назад через матрицю, починаючи з нижнього правого кута. Якщо безпосередньо над, або безпосередньо праворуч комірки таблиці міститься значення, що дорівнює значенню в поточній комірці, то потрібно переміститися до цієї комірки (якщо обидві комірки таблиці задовольняють умову, то вибирається одна з них). Якщо обидві такі комірки мають значення, строго менше значення в поточній комірці, то потрібно переміститися по діагоналі вгору-вліво, і вивести символ. Такий алгоритм виведе символи в LCS у зворотному порядку. Наприклад, на матриці вище, це буде DABA.

Даний алгоритм має назву «динамічне програмування знизу вгору». Ось інший спосіб динамічного програмування, його називають «динамічне програмування зверху вниз».

Основна ідея: є рекурсивний алгоритм для певної проблеми, що дає дійсно велику кількість повторень, таку як:

$$T(n) = 2T(n - 1) + n. \quad (3.4)$$

Однак багато з підзадач, які вирішуються, при рекурсивному проходженні дерева вниз, однакові. Тоді можна зменшити кількість ітерацій алгоритму, якщо зберігати обчислення так, щоб обчислювати кожну різну підзадачу один раз. Ці рішення можна зберігати в масиві або хеш-таблиці. Такий погляд на динамічне програмування часто називають мемоаудіювання. Наприклад, для проблеми LCS було створено наступний рекурсивний алгоритм:

```
f (S, k, T, l)
{
    if (k==0 || d ==0)  {
    return 0;
}
if (S[k] == T[d])
{
    result = 1 + f(S, k-1, T, d-1);
}
else {
    result = max( f (S, k -1, T, d),  f(S, k, T, d-1) );
}
    return result;
}
```

Цей алгоритм виконується за експоненційний час. Насправді, якщо S і T мають повністю непересічні множини символів (так що ми ніколи не виконується S[k] == T [d]), то кількість рекурсивних викликів буде надто великою. У мемоїзованій версії результати в матриці зберігаються так, щоб будь-який набір аргументів LCS створював нові рекурсивні виклики тільки один раз. Мемоїзована версія починається з ініціалізації Arr[i][j] невідомого для всіх i, j, а потім продовжується наступним чином:

```
f(S, k, T, d)
{
    if (k==0 || d==0) {
        return 0;
    }
    if (Arr[k][d] != undefined) {
    return Arr[n][m]  // (*)
    };
    if (S[k] == T[d]) {
    result = 1 + f(S, k-1, T, d-1);
    }
    else {
        result = max( f(S, k-1, T, d), f(S, k-1, T, d-1) );
        }
        Arr[k][d] = result; // (**)
        return result;
}
```

Порівнюючи з попередньою версією алгоритму, було змінено збереження поточних обрахунків в рядку (**), нові рекурсивні виклики в даній версії алгоритму створюються, тільки якщо відповідь у рядку (*) ще не була обрахована.

У цій версії алгортму час роботи O(k*d). Алгоритм виконує рядок (**) не більше одного разу для будь-якого заданого значення параметрів. Це означає, що кількість рекурсивних викликів загалом не перевищує 2*k*d. Будь-який заданий виклик функції передбачає лише O(1) (виконання перевірок рівності і обрахування max, або додавання 1), тому загальний час роботи O(k*d).

Порівнюючи динамічне програмування «знизу-вгору» і «зверху-вниз», можна зробити висновок, що обидва алгоритми працюють майже однаково. Версія згори-вниз (мемоїзована) продукує рекурсивні виклики, але потенційно може бути продуктивніше, ніж версія «знизу-вгору», у ситуаціях, коли деякі з підзадач ніколи не обчислюються. Ці відмінності, однак, є незначними. Тож для даної задачі була використана перша версія алгоритму, яка є найпростішою і найбільш інтуїтивною для вирішення даної проблеми.

3.5. Порівняння акустичних відбитків аудіофайлів

Для порівняння акустичних відбитків аудіофайлів був використаний алгоритм MinHash.

Алгоритм MinHash є апроксимацією коефіцієнту Жаккара для двох множин.

Коефіцієнт Жаккара дорівнює нулю, коли множини не мають спільних елементів, і одиниці, коли множини рівні. Розрахунок кофіцієнту Жаккара наведений у формулі (3.5).

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}, \tag{3.5}$$

де A, B – множини, для яких обчислюється коефіцієнт Жаккара.

Кількість порівнянь для N кількості множин наведена у формулі (3.6), це наближення еквівалентне порівнянню кожної пари множин тільки один раз, без необхідності порівняння кожної множини з самою собою.

$$k=\frac{N(N-1)}{2} \approx \frac{N^2}{2} \tag{3.6}$$

Така кількість порівнянь вимагає потужних обчислювальних ресурсів. Для зменшення потреб в обчислювальних ресурсах було використано алгоритм MinHash.

Для кожного набору даних було розраховано сигнатуру MinHash. Сигнатури MinHash мають фіксовану довжину, для заданої максимальної величини похибки, незалежно від розміру множини. Кількість необхідних хеш-функцій, для заданої максимальної величини похибки наведена в формулі (3.7).

$$k = \left\lceil \frac{1}{\varepsilon^2} \right\rceil, \tag{3.7}$$

де ε – максимальна величина похибки.

Для апроксимації коефіцієнту Жаккара для двох множин, потрібно обрахувати кількість співпадаючих хеш-значень в сигнатурах даних множин.

```
class MinHash(object):

    def __init__(self, k, seed=10):

        self._k = k
        self._seed = seed

        minint = np.iinfo(np.int64).min
        maxint = np.iinfo(np.int64).max

        self._masks = (np.random.RandomState(seed=self._seed)
                       .randint(minint, maxint, self._k))

        self._hashes = np.empty(self._k, dtype=np.int64)
        self._hashes.fill(maxint)

    def add(self, v):

        hashes = np.bitwise_xor(self._masks, hash(v))

        self._hashes = np.minimum(self._hashes, hashes)

    def jaccard(self, other):

        if np.any(self.masks != other._masks):
            raise Exception('Can only calculate similarity '
                            'between MinHashes with the same hash '
                            'functions.')

        return (self._hashes == other._hashes).sum() / float(self._k)
```

Рисунок 3.5 – Функція обчислення MinHash

Алгоритм MinHash є апроксимацією коефіцієнту Жаккара для двох множин.

3.6 Опис інтерфейсу користувача

Система має зручний та інтуїтивно зрозумілий користувацький інтерфейс. На рис. 3.6 зображена початкова сторінка системи.

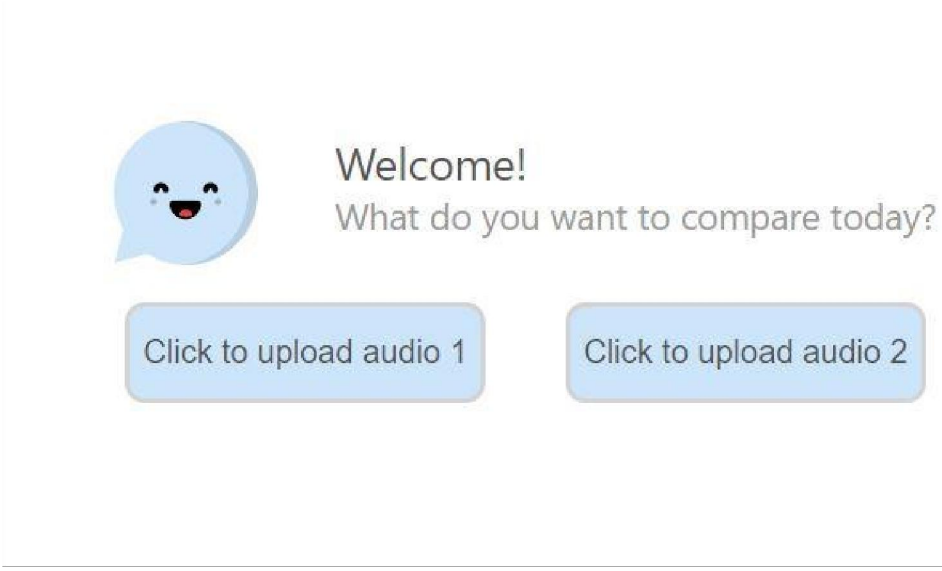

Рисунок 3.6 – Початкова сторінка системи

Наступним кроком користувач може завантажити аудіофайли з файлової системи свого косп'ютера для їх порівняння. Після натискання користувача на кнопку завантаження файлу з'являється вікно вибору файлу.

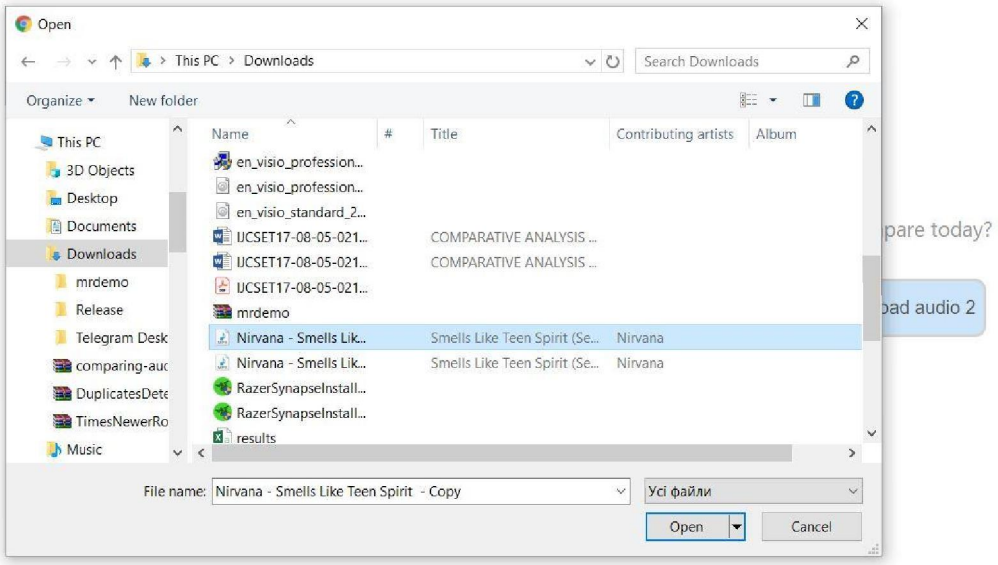

Рисунок 3.7 – Вікно вибору файлу

У випадку вибору користувачем файлу, що не відповідає MP3 формату, буде виведено відповідне повідомлення, файл у систему завантажуватись не буде.

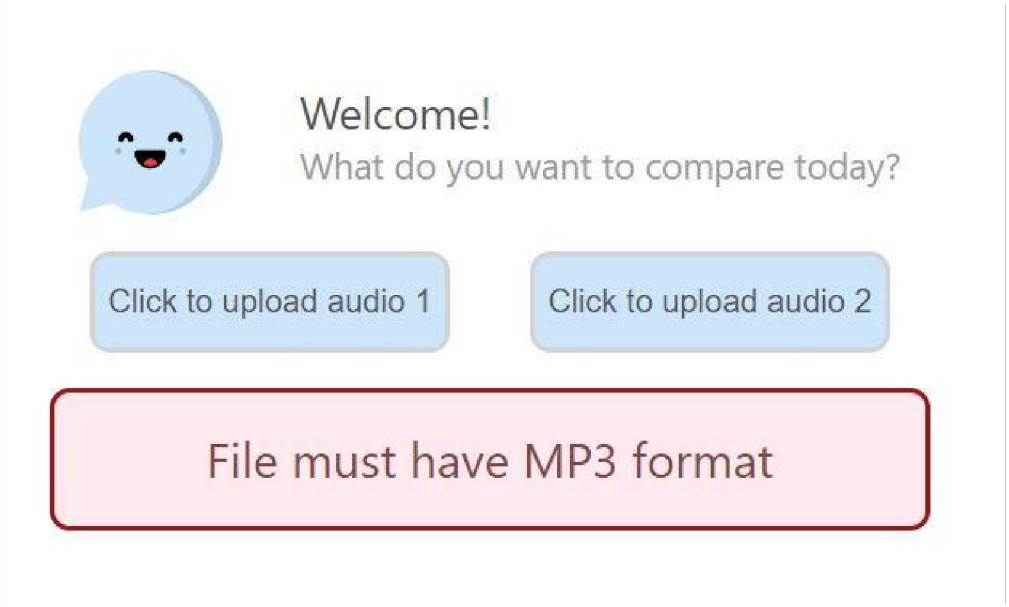

Рисунок 3.8 – Повідомлення про невідповідний формат файлу

Після успішного завантаження двох аудіофайлів користувач має можливість їх порівняння.

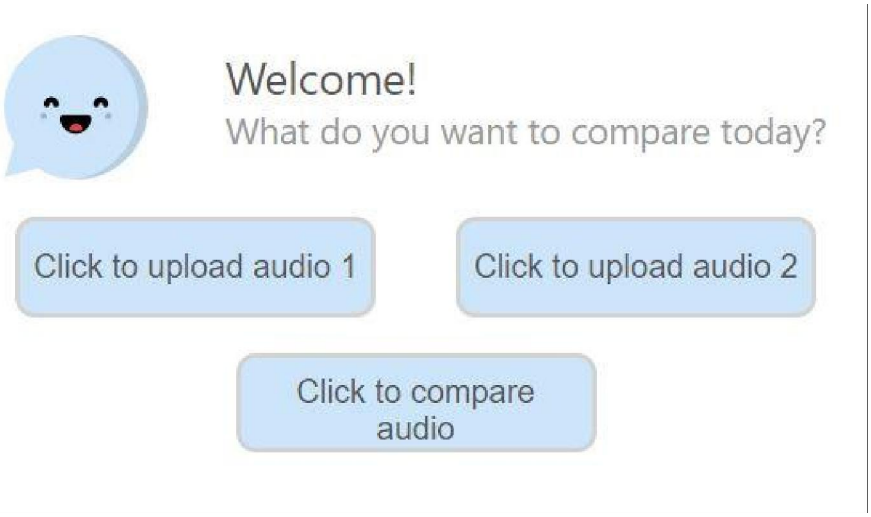

Рисунок 3.9 – Сторінка для порівняння аудіофайлів

Після натискання кнопки “Click to compare audio” виводяться результати порівняння завантижених користувачем аудіофайлів.

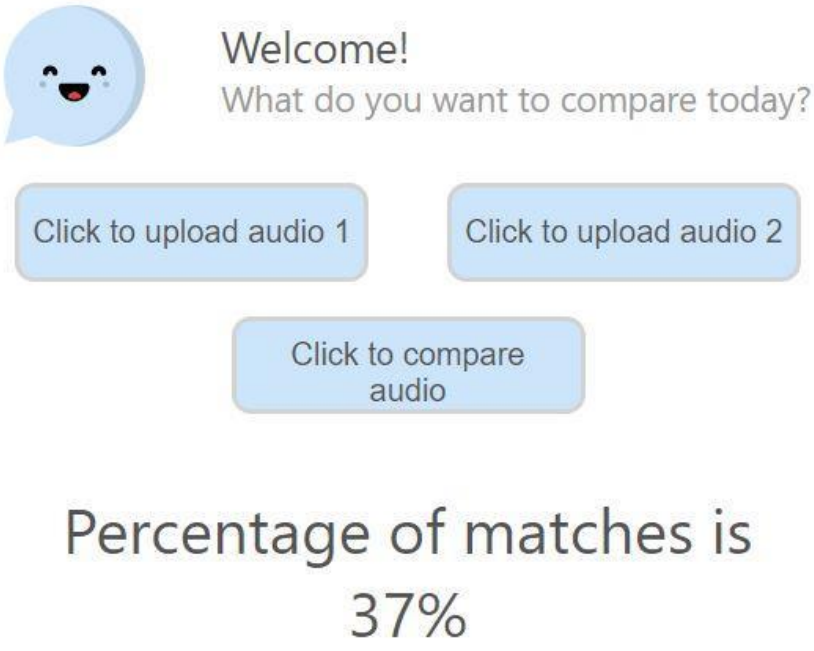

Рисунок 3.10 – Результат порівняння аудіофайлів

ВИСНОВКИ

Метою даного дипломного проекту було створення веб-орієнтованої системи для визначення збігів в аудіофайлах.

В результаті аналізу існуючих рішень було визначено їх основний недолік: можливість ідентифікації лише тих аудіофайлів, які присутні в базі даних додатку.

В процесі роботи було вивчено теоретичне підґрунтя декодування аудіофайлів та отримання частотних характеристик аудіосигналу, виділення характерних частот аудіосигналу, створення акустичних відбитків аудіотреків.

При розробці веб-орієнтованої системи для визначення збігів в аудіофайлах використані сучасні технології та новітні рішення обробки аудіофайлів.

Аналіз способів ідентифікації аудіофайлів показав доцільність використання методу акустичного відбитку аудіофайлу.

В якості програмних засобів для реалізації системи було використано мову програмування JavaScript та бібліотеку React з її інфраструктурними компонентами, що дозволило створити зручний інтерфейс користувача та забезпечити підтримку популярних браузерів. Серверна частина додатку була створена з допомогою мікрофреймворку Flask та мови програмування Python.

Завдяки використанню алгоритму декодування MP3 файлів, система забезпечує можливість завантаження та обробки стиснених звукових файлів.

Розроблена система визначає відсоток збігів в аудіофайлах за допомогою алгоритму MinHash для множин, користуючись представленням акустичного відбитку аудіосигналу у вигляді множини хеш-функцій.

Система має потенціал подальшого розширення функціоналу та можливість роботи в режимі on-line. У подальшому програмний інтерфейс системи може бути використаний сервісами потокової музики для уникнення дублікації аудіоінформації, сервісами з публічним доступом до аудіоінформації для роботи зі скаргами власників контенту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. How does Shazam work? Music Recognition Algorithms, Fingerprinting, and Processing [Електронний ресурс] – Режим доступу до ресурсу: https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition.
2. Как работает Now Playing в Google Pixel 2 [Електронний ресурс] – Режим доступу до ресурсу: https://myachinqa.blogspot.com/2018/01/now-playing-google-pixel-2.html.
3. Цифровий водяний знак [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B8%D0%B9_%D0%B2%D0%BE%D0%B4%D1%8F%D0%BD%D0%B8%D0%B9_%D0%B7%D0%BD%D0%B0%D0%BA.
4. Basic concepts behind Web Audio API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/enUS/docs/Web/API/Web_Audio_API/Basic_concepts_behind_Web_Audio_API.
5. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript.
6. Acoustic fingerprint [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Acoustic_fingerprint.
7. Вступ до JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://ua.freelook.info/d/21-vstup-do-javascript.
8. Practical Introduction to Frequency-Domain Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://www.mathworks.com/help/signal/examples/practical-introduction-to-frequency-domain-analysis.html.

9. Python [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Python.
10. Audio Fingerprinting with Python and Numpy [Електронний ресурс] – Режим доступу до ресурсу: https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/.
11. Flask [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Flask.
12. Патент США: № US5918223A, МПК G 06 F 17/30. Method and Article of Manu-facture for Content-Based Analysis, Storage, Retrieval, and Segmentation of Audio Information., 22.06.96.
13. Kim Hyoung-Gook,Moreau Nicolas,Thomas Sikora, MPEG-7 Audio and Beyond: Audio Content Indexing and Retrieval, JohnWiley & Sons, 2005.
14. Low-order auditory Zernike moment: a novel approach for robust music identification in the compressed domain [Електронний ресурс] – Режим доступу до ресурсу: https://asp-eurasipjournals.springeropen.com/articles/10.1186/1687-6180-2013-132.
15. Звукові хвилі [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/kolivannaihvilijk/home/zvukovi-hvili.
16. MPEG-1 [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/MPEG-1.
17. Цифрова музика не могла існувати без перетворення Фур'є [Електронний ресурс] – Режим доступу до ресурсу: https://uk.upost.info/31363939313535323837.
18. K. Kondo, “Method of changing tempo and pitch of audio by digital signal processing.” Google Patents, 1999.